DATABASE DESIGN II - 1DL400

Spring 2014
2014-01-30

A course on modern database systems

http://www.it.uu.se/research/group/udbl/kurser/DBII_VT14/activedb.pdf

Tore Risch
Uppsala Database Laboratory
Department of Information Technology, Uppsala University,
Uppsala, Sweden

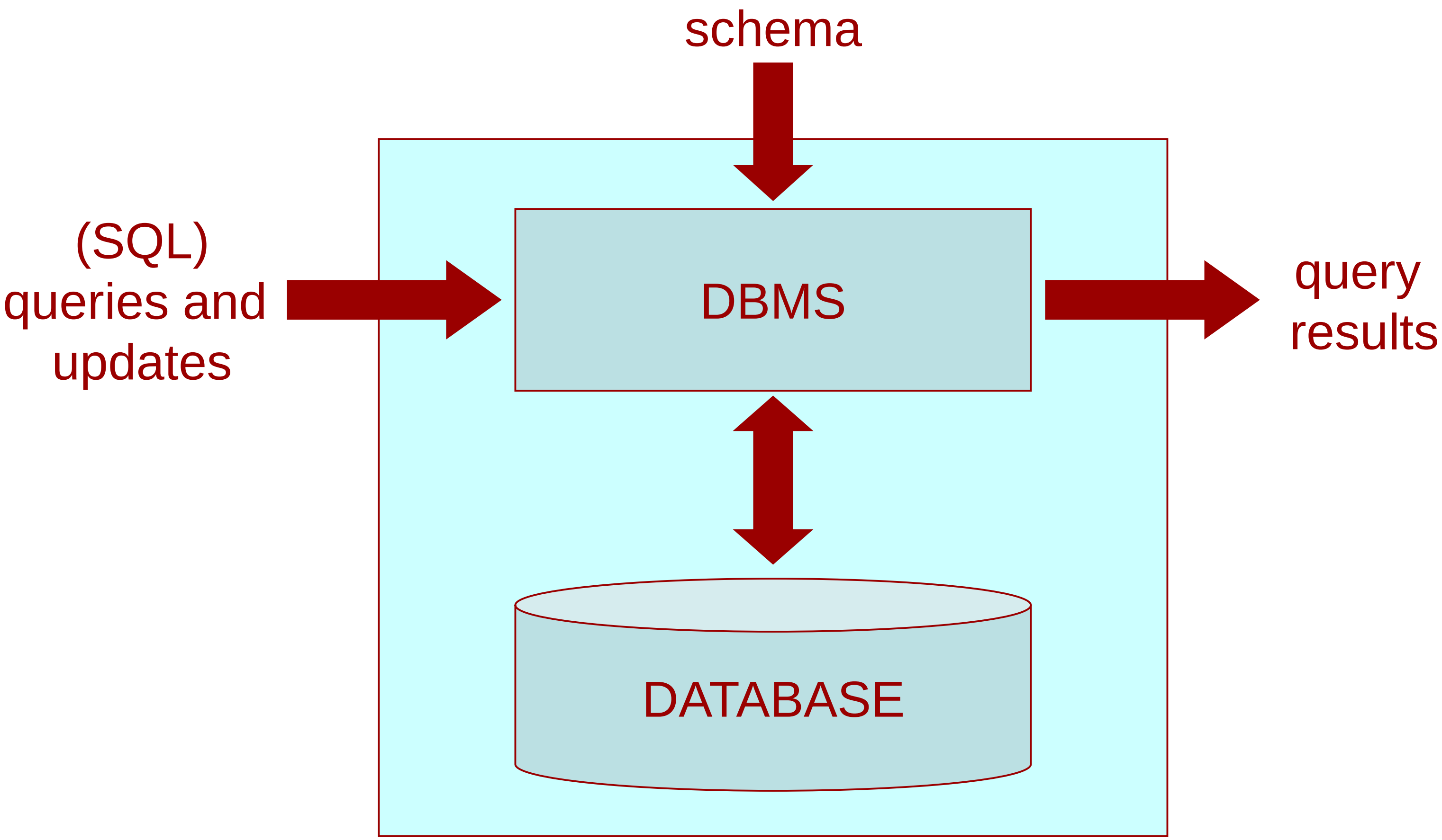
Active Databases

Elmasri/Navathe ch 24.1
Padron-McCarthy/Risch ch 15

Tore Risch
Uppsala Database Laboratory
Department of Information Technology, Uppsala University, Uppsala, Sweden

Active Databases

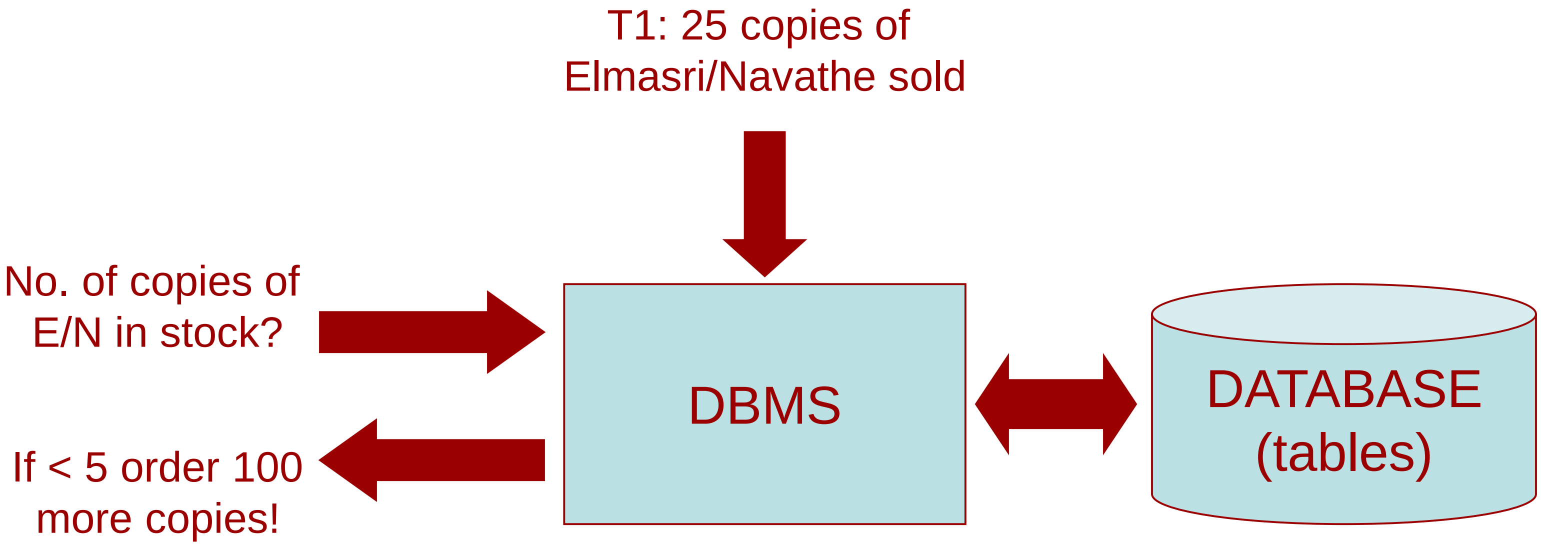
General principles of conventional DBMSs



Conventional (Passive) DBMSs

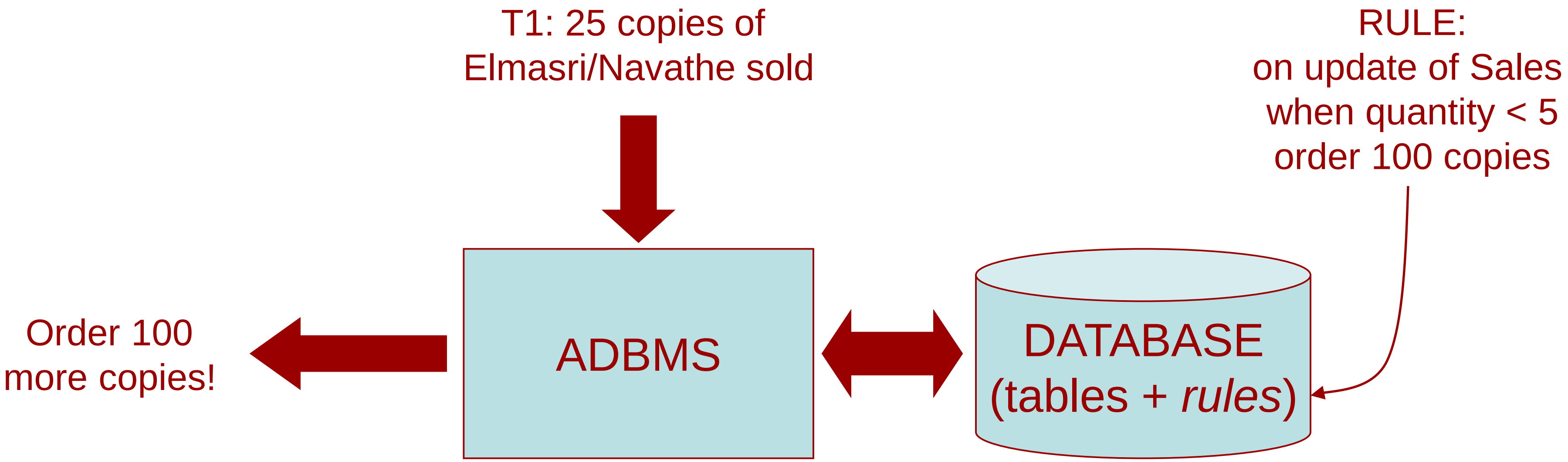
- Provides data model (e.g. the relational data model)
- Provide transaction model
  - ACID principle, e.g. updating account info, short transactions, small updates
  - *Passive* model because client controls database updates
- Examples of real world problems not so well suited for passive databases:
  - Inventory control
    - reordering of items when quantity in stock falls below threshold.
  - Travel waiting list
    - book ticket as soon as right kind is available
  - Stock market
    - buy/sell stocks when price below/above threshold
  - Maintenance of master tables, *view materialization*
    - E.g. maintain table that contain sum of salaries for each department

Conventional Passive DBMS Solution


T1: 25 copies of Elmasri/Navathe sold
No. of copies of E/N in stock?
If < 5 order 100 more copies!
DBMS
DATABASE (tables)

- In a *passive* database system, the application will periodically *poll* the DBMS:
  - Frequent polling => expensive
  - Infrequent polling => might miss the right time to react
  - The problem is that the DBMS *does not know* that application is polling

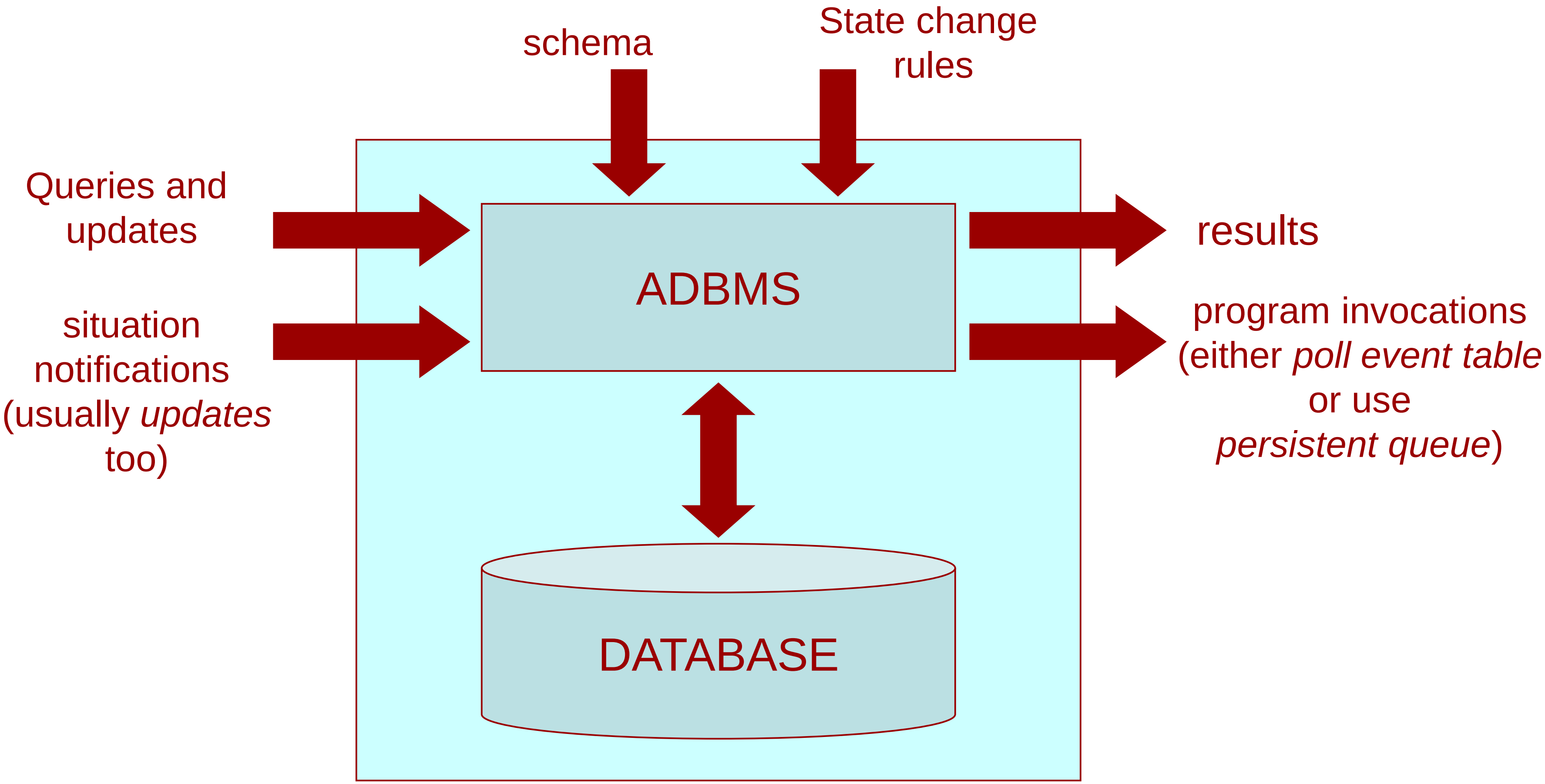
Active Database Solution


T1: 25 copies of Elmasri/Navathe sold
RULE: on update of Sales when quantity < 5 order 100 copies
Order 100 more copies!
ADBMS
DATABASE (tables + *rules*)

- In an *active* database system, the ADBMS recognizes predefined *situations* (i.e. state changes) in the database .
- The ADBMS *triggers* predefined *actions* when situations occur, typically database updates or calls to stored procedures.
- Actions are usually database updates, not calls to external programs to order items as in the example.

Active Database Management System

- The general idea is that an ADBMS provides regular DBMS primitives and in addition state change rules called *triggers*:
  - + defining application-defined situations identifying state changes
  - + triggering application-defined reactions when state changes occur


schema
State change rules
Queries and updates
situation notifications (usually *updates* too)
ADBMS
results
program invocations (either *poll event table* or use *persistent queue*)
DATABASE

Applications for active databases

- Notification
  - Automatic notification when certain condition occurs
  - Oracle provides *persistent queue* of program invocations
  - If not supported => poll event table
- Enforcing integrity constraints
  - Triggers are on a lower programming level than database *constraints* (explained later)
  - Can identify state changing *situations,*
- Maintenance of derived data
  - Automatically update derived data (materialized views) to avoid anomalies due to redundancy

Active database rule models

- Event-Condition-Action (ECA) rules is the most common model.
  - Semantics of ECA rules:
    - WHEN *event* occurs - IF *condition* holds - DO execute *action*
  - Event:
    - Usually an update of database record(s)
    - Parameterized by using pseudo tables named OLD containing table state *before* the update, and NEW containing the table state *after* the update.
  - Condition:
    - Query on database old and new database state as database queries
    - Condition is considered true if query returns non-empty result
  - Action:
    - Usually SQL update statements or call to stored procedure referencing the updated row(s)

EA trigger example

Example of EA (Event – Action) trigger for maintaining derived attribute *department.totalsal* attribute in tables:

```
employee(ssn, salary, dno)
department(dno, totalsal)

create trigger totalsal1
   after update on employee
   for each row
   begin update department
               set totalsal = totalsal + new.salary
               where dno = new.dno;
         update department
               set totalsal = totalsal - old.salary
               where dno = old.dno;
   end;
```

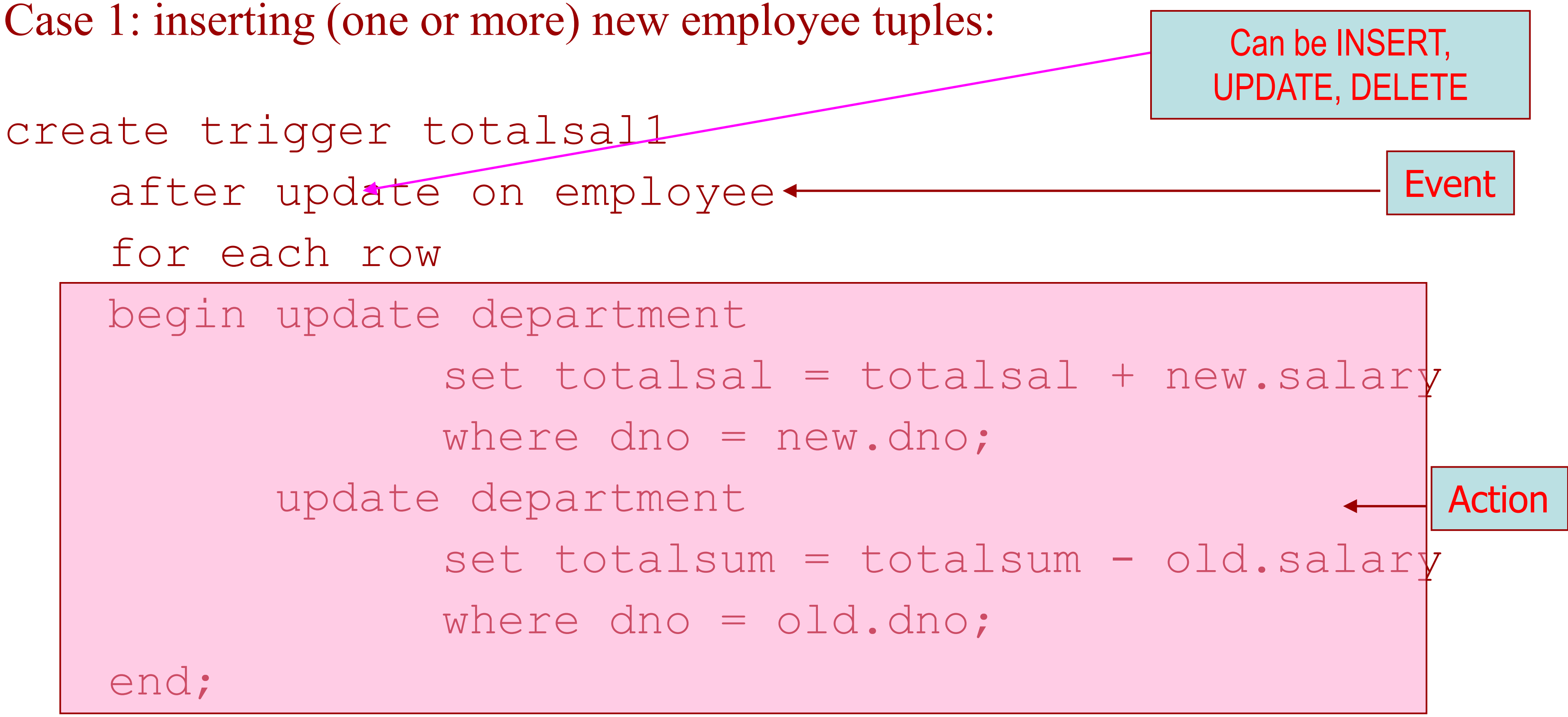
Notice: ADBMS sees update as delete followed by insert

EA trigger example

Employee and department tables:

```
employee(ssn, salary, dno)
department(dno, totalsal)
```

Case 1: inserting (one or more) new employee tuples:



```
create trigger totalsal1
    after update on employee
    for each row
    begin update department
                set totalsal = totalsal + new.salary
                where dno = new.dno;
          update department
                set totalsum = totalsum - old.salary
                where dno = old.dno;
    end;
```

Can be INSERT, UPDATE, DELETE

Event

Action

EA trigger example cont …

Database state change case analysis should be done:

1. Does it work if someone is hired?
2. Does it work if someone is fired?
3. Does it work if someone changes department?
4. Does it work if a department is deleted?
5. Does it work is a new department is created?
6. Are these all possible state changes?

Eventually more triggers are needed!

Question: Are more triggers needed in this example?

Row-level vs. statement-level triggers

- Triggers can be:
  - **Row-level**
    - FOR EACH ROW specifies a row-level trigger
  - **Statement-level**
    - FOR EACH STATEMENT (default when FOR EACH ROW is not specified)
      -
- Row level triggers
  - Executed separately for each row affected for a given SQL statement (usually update)
- Statement-level triggers
  - Executed only once per entire SQL (update) statement sent to the DBMS
  - Makes difference when update over many rows specified in update statement

Non-procedural alternative: Materialized views

Modern DBMSs (e.g. Oracle) has *materialized views*:

```
create materialized view department
as select dno, sum(salary) as totalsal
   from employee
```

- A *regular view* is a virtual table, which is not stored in the database but computed when a query using the table is issued.
- By contrast a *materialized view* is master table, which is *automatically* maintained by the DBMS when there are updates on any of the tables in its view definition.
- Here: *department* automatically updated when *employee* is updated.
- Materialized views are not standard: DBMS may not have it, syntax may differ.
- Check manual for *efficiency* of materialized view maintenance.

ECA trigger example

Example of ECA (Event – Condition - Action) trigger for maintaining salary constraint that the boss always earns more:

```
employee(ssn, salary, dno)
department(dno, mgrssn)
```

Situation1: Check employee salary increases

```
create trigger employee_raise
    after update of salary on employee
    for each row
    when (select * from employee m, department d, new
          where new.dno = d.dno and
                new.ssn <> d.mgrssn and
                            new.salary > m.salary and
                m.ssn = d.mgrssn and
                m.dno = d.dno)
    begin update employee e
                set salary = old.salary*0.9
                from orow; end;
```

new is employee in d

new gets higher salary than d:s boss

lower new salary

ECA trigger example

Example of ECA (Event – Condition - Action) trigger for maintaining salary constraint that the boss earns more:

```
employee(ssn, salary, dno)
department(dno, mgrssn)
```

Situation 2: Check boss salary

```
create trigger boss_salary
   after update of salary on employee
   for each row
   when(select * from employee e, department d, new
        where new.dno = d.dno and
              new.ssn = d.mgrssn and
              new.salary < e.salary and
              e.dno = d.dno and
              e.ssn <> d.mgrssn)
   begin rollback; end;
```

new is boss in d

Some employee e in d has higher salary

ECA trigger example cont …

Database state change case analysis should be done here too:

1. Does it work if someone is hired?
2. Does it work if someone is fired?
3. Does it work if someone changes department?
4. Does it work if a department is deleted?
5. Does it work if a manager is hired?
6. Does it work is a manager is fired?
7. Does it work if a manager’s salary is lowered?
8. Does it work if an employee becomes a manager?
9. Does it work if a manager becomes an employee?

Any more situations?

Question: What more triggers needed in this example?

Non-procedural alternative: Assertions

Modern DBMSs have *assertions*:

```
create assertion salary_constraint
    check (not exists
           (select *
            from employee e, employee m, department d
            where e.salary > m.salary and
                  e.dno = d.dno and
                  d.mgsssn = m.ssn))
```

- Implementation of assertions (triggers, stored procedures) may differ in different DBMSs.
- For example, advanced assertions may not be supported by the DBMS or be very inefficient.
  - A naive implementation of assertions that checks constraint after each update *does not scale*.
- Assertions cannot make *compensating actions* depending on situation as triggers

Rule variants

- EA – Even Action rules
  - Condition always true as in our first example
- CA – Condition Action rules
  - Event detected by system
  - Common in AI, forward chaining systems, OPS5 programming language
  - Usually not in databases
  - Difficult to identify actual state changes
- A – Action
  - Would be stored procedures
- C – Condition
  - Would be assertions

Summary active databases

- Active DBMSs provide situation-action rules in database
- Supports many functionalities: e.g. integrity control, derived data, change notification, monitoring, database replication
- Cautions:
  - very powerful mechanism:
  - small statement => massive behavior changes.
  - rope for programmer.
  - requires careful design and situation analysis
- Make state change case analyzes when designing triggers.
  - Make sure indefinite triggering or *undesired cascading triggering* cannot happen.
- *Avoid using triggers* unless really needed.
  - Use queries, view materialization statements, referential integrity constraints, or stored procedures instead if possible.
- DBMS itself uses triggers a lot
  - E.g. data replication and constraint management in Oracle